Chapter 14: ecommerce

by Philip Greenspun, part of Philip and Alex's Guide to Web Publishing

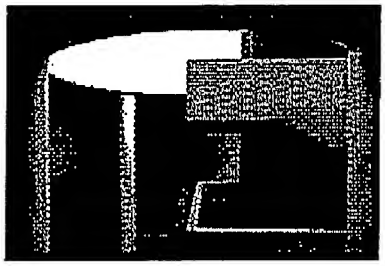
"Electronic commerce" conjures up visions of product catalog sites and shopping baskets, things that were passé back in 1995. Why then a chapter on ecommerce?

First, because so few people are doing it right. A friend of mine said that he wouldn't buy an airplane ticket from the United Airlines Web site. He wasn't paranoid about his credit card number being compromised. Drawing an inference from their sluggish and badly programmed Web service, he didn't believe that his reservation would be made or a ticket issued.

Second, because people are spending obscene amounts of money to do trivial things. Sometimes the money is spent on buying packaged junkware and then on figuring out how to debug/integrate it. Sometimes the money is spent on madly writing, compiling, and debugging Java programs that replicate capabilities already present in sets of Perl or Tcl scripts three years ago.

Finally, because even a bland ecommerce site may occasionally blossom into an interesting site with community or Web service angles.

Step 1: Decide if You're Serious and Responsible



Curious to see if some intervening months had sufficed for United Airlines to get their programming act together, I tried to buy a ticket on-line on April 22, 1998. Even at 2:00 am, the site was sluggish, to be sure, but I managed to find a $300 San Francisco/Boston round-trip. When I tried to buy the ticket, I got a server error message with instructions to try later. A few hours later, the site was operational but the $300 fare was no longer listed. I called United and they said they'd be happy to sell me a ticket, but it would be $1900. The $300 fare had only been available earlier in the day. When I noted that the only thing that had prevented me from buying the ticket was their broken site and would they consider honoring the fare, they said I should call their customer relations number, which I did. I reached a recording that said "32-10 you have dialed an invalid number".

Ultimately I did reach customer relations where they also refused to honor the fare, mumbling something about how the Web was just an experimental thing anyway. In one transaction, United had managed to simultaneously demonstrate incompetence and unwillingness to take responsibility for its incompetence.

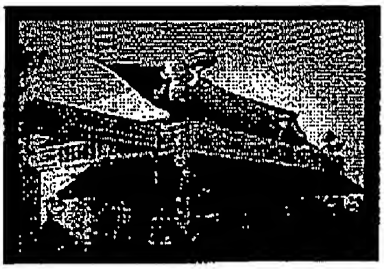
Can you learn from United's mistakes? Yes. Before you set up an ecommerce site, decide (1) whether you are serious about building a working service, and (2) whether you are going to take responsibility for your programming and administration mistakes.

Step 2: Think About Your Accounting System

Though one of my leitmotifs is "packaged software probably won't solve your problem", accounting software is an exception. Web services tend to be new and varied. It would be a hopeless fantasy for me to go down to CompUSA and try to buy a birthday reminder server that accomplishes what the custom Tcl/SQL scripts at http://www.arsdigita.com/remindme/ do. But if ArsDigita, LLC wanted to keep track of invoices, payments, and taxes, it would be insane to write an accounting system from scratch. Our Web publishing goals aren't shared by too many other people but our accounting problem is the same as that of any other business.

Thus at the end of the day, you're going to have a custom-developed Web application talking to a standard accounting system of some sort. If the accounting system is powerful and running out of the same RDBMS installation as the Web site, then you might well not need to confront any of the allegedly tough ecommerce technology decisions.

Step 3: Where Do You Park the Database?



In the past four years, a particular kind of ecommerce site development project seems to cross my desk repeatedly. A manufacturing company has been selling to retailers for decades. The Web site will be the first time that they've sold directly to customers. Customers will get the ability to order semi-custom products and the factory will get real-time access to information about what customers want.

An obvious first cut at the problem involves three database installations:

1. an RDBMS on the Web server to hold catalog info, user personalization information, and orders
2. an RDBMS at a bank or service bureau that holds credit card numbers to be billed
3. an RDBMS in the factory that collects orders in batches from the Web server

There are some good things about this solution. You aren't responsible for keeping customer credit card numbers secure. Data that are needed for real-time decisions are kept close to the programs that use the data.

Suppose, however, that the Web server needs a lot of data from the factory tables before it can offer delivery information. It is possible to keep copies of the necessary factory tables in the Web server's RDBMS. Every time an update or insert is made on the factory RDBMS, the transaction is duplicated on the Web server RDBMS. This is called *database replication* and companies like Oracle produce software to facilitate replication. However, you have to budget extra time and money to develop and maintain a replication strategy. So you might consider eliminating the RDBMS next to the Web server and configuring the Web server program to make an encrypted connection to the factory RDBMS.

Suppose that you are quickly turning out custom goods. You can't wait for batches of orders to be sent to the factory because products are supposed to be built and shipped on the same day as orders placed before 2:00 pm. This means that the factory RDBMS must have rapid access to the relevant tables from the Web server RDBMS. Again, you have the option of replication or eliminating the factory database installation and having the factory client programs talk directly to

the Web server RDBMS.

Suppose that you want to offer customers the option to "bill the same credit card as on your last order, ending in 4561." In that case you either need to keep the old credit card numbers yourself or have some means of communication between your database and your transaction vendor (bank).

Suppose that your accounting system can run the credit card numbers. In that case, you will probably want to eliminate the RDBMS at the bank and tightly couple the Web site to whatever RDBMS installation is running the accounting system.

There are no correct answers to the questions of "how many RDBMS installations and where are they" but you need to address them nonetheless.

Step 4: Lifetime Customer Value Management

Companies that are selling direct for the first time will immediately realize that they now have access to their dream data. They can figure out, down to the individual consumer, who is buying what and how often. About two days after the business folks realize this, they will turn to the nerds and say "Build us a lifetime customer value management system."

The "lifetime" in the above phrase reflects the fact that you have to engineer the system to track particular customers' activities over many years. As soon as Joe Smith shows up on the site, you need to know how many times this person has placed an order, returned a product, demanded a refund, come to the site and not ordered, etc.

The engineering in a system like this is a straightforward matter of SQL tables and Web scripts. What's hard are the business rules. Here are some examples:

- Rachael Supergreat has ordered 10 times before. Though we're running low on inventory, push her order to the front of the list and quote her a 24-hour delivery time.
- Tristan Pretty Good has ordered once before. However, it was a year ago so he's probably not going to turn into the best possible customer. If the factory is behind schedule, quote him a three-week delivery time and see if he still wants to place the order.
- Phoebe Unknown has never ordered before. She might become a very valuable customer, so give her order medium priority and quote her a one-week delivery time. We don't want to risk turning her away.
- David Difficult has ordered custom products eight times before and returned them each time. He did not include the original packing material on six of those returns and damaged the product cosmetically three times. Give him a "server busy, please try again later" page.

As a Web developer, I don't think there are customer value management *technical* issues that are distinct from those associated with personalization on non-commercial sites. But as a publisher or service designer, you should at least be aware that you'll have to spend a few weeks thinking about what rules to implement.

Case Study: MIT Press

MIT Press is a publisher of books and academic journals. Since 1993, they'd been putting progressively more catalog info and

content on their Web site, which was developed and maintained by Terry Ehling. She originally worked part-time but by 1996 being webmistress was a full-time job. In 1996, Terry came to me with the following goals for her Web site (http://mitpress.mit.edu):

- Serve up a catalog of 6000 books and 40 journals in a consistent fashion; the existing collection of static files was becoming too cumbersome to maintain.
- Collect comments on the catalog items.
- Collect names and contact information for readers who wished to be on various mailing lists.
- Collect orders for products.
- Allow MIT Press personnel to spam groups of readers with announcements, e.g., "those readers who've signed up to the cognitive science mailing list".

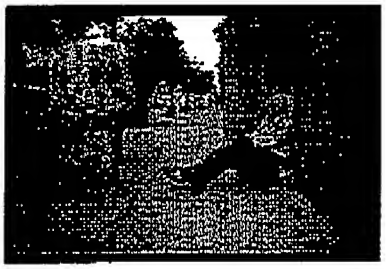
Decision 1: How Much to Change the Business?

MIT Press maintained overlapping catalog data in several separate systems. Their accounting/ordering system was a turnkey software package for academic publishers. It was coded originally for the Data General Nova 16-bit minicomputer. As these computers were no longer manufactured, the software was running on an IBM PC emulating the old Nova instruction set. To someone raised in the Oracle/Unix tradition, this sounds kind of insane but in fact nearly all academic publishers run the same software. None of them do enough volume to justify developing a replacement package. Anyway, this turnkey system had some information on each book. The production folks had a FoxPro database with information on current and past projects. The legacy Web site had pretty good information on the last few years' worth of books but these data were merely arranged in Unix filesystem files.

Installing and maintaining an RDBMS is a fair amount of work. If we were going to all that trouble to make the Web service work, it seemed like a good idea to eventually run production and accounting from the same database. Of course, what seems like a good idea to an engineer doesn't always seem like a good idea to a business person. The management of the Press wasn't ready to consider fundamental changes to the rest of the business.

One side effect of this decision was that it got us out of the on-line credit card verification business. We'd just have to pipe credit card numbers along with orders to the existing accounting system and let those folks handle them the way they handled other credit card orders. If a customer gave us a bad number, we'd email him.

Decision 2: Shopping Baskets?



A primary purpose of the Web site was to facilitate on-line shopping. Everyone else on the Internet at the time was implementing shopping basket systems. I had built some for clients in 1995 but by 1996 had soured on the idea. It wouldn't have been a horrible concept if every site used the same shopping cart system. But each site had its own user interface. Why should you have to learn how to "check out" when all you wanted was one book?

I argued for an invisible shopping cart. A user would click on a product and get an order form. After submitting it with shipping address and credit card number, a magic cookie would be written back to his browser. The user would be offered the opportunity to continue shopping. If he clicked on an "order" link again, he'd be offered the opportunity to add the item to his previous order. The system would degrade gracefully. If the user's browser didn't adhere to the Netscape magic cookie spec, he'd still be able to place orders for products, but it

would be a little more cumbersome. If an order had already been transmitted to the old Data General Nova system then the user wouldn't be offered the "add to old order" option. If the user quit his browser and restarted, he wouldn't get the "add to old order" option.

We decided to go for the invisible cart. Orders initiated the previous day would be transmitted in a batch from the RDBMS to the legacy system every morning at 7:00 am. That meant a user had between 7 and 31 hours to complete an order using the invisible cart.

Choosing Technology

We elected to use AOLserver talking to the Illustra database. MIT Press had a puny SPARC 5 Web server and had not budgeted for a larger machine, yet we knew that we'd have to serve about 200,000 hits a day from this computer, some of them requiring RDBMS queries. I knew from bitter experience that Illustra would not scale very well and that it was likely to periodically wedge itself. However, I had a fairly good sized library of AOLserver Tcl scripts with embedded Illustra queries, the PLS Blade for Illustra was a brilliantly useful tool, and MIT Press did not have the budget to install or maintain an enterprise-class RDBMS such as Oracle.

Moving the legacy data

It turns out that working in the publishing world is a programmer's dream. The industry has been forced for many years to key everything by the International Standard Book Number (ISBN). So there are very few thorny data modeling issues.

To avoid typing 6000 book titles, we got a text file dump of the data in the Data General Nova system. Then we wrote some Perl scripts to convert these into SQL inserts and fed the whole batch to Illustra's Unix shell client. We wrote some AOLserver Tcl scripts to semi-automatically transfer over book jacket images and long descriptions from files on the old Web server. We decided not to store book and journal images in the RDBMS. Instead, we kept a Unix file system directory for each catalog item. For books, the AOLserver Tcl script would look in the directory for `cover-sm.gif` or `cover-sm.jpg`. If present, it would be displayed on the book catalog page (see http://mitpress.mit.edu/book-home.tcl?isbn=0262650398). If not, we'd serve a page with a slightly different design (see http://mitpress.mit.edu/book-home.tcl?isbn=0262560631).

Integrating the Graphics

MIT Press brought Ben Williams (www.blueperiod.com) up from New York to do the graphic design. I showed him that each book page had the following computational structure:

1. Query the `books` table for basic title and description for the ISBN key in the URL.
2. Stream out the top part of the catalog page.
3. Query the `reader_comments` table to pull out reader comments for display right on the page.
4. Stream out the reader comments, if any.
5. Query the `books` table again to find related items, such as a paperbound version of the same title.
6. Stream out the related items (as hyperlinks).
7. Do an expensive LIKE query against the `books` table to find other books by the same author

(s).
8. If found, stream out the list of books.

I explained that, of the four queries, the last one would take the longest. If we wanted to have a site that felt responsive, we'd absolutely have to avoid making the entire page an HTML table. The user shouldn't notice that the LIKE query was taking a second or two because he'd be reading the top part of the page.

Ben eventually figured it all out and designed a beautiful looking site with stacked tables. The first table contained the information from the first three RDBMS queries and the last contained the "books by the same author" content plus the page footer. It was fast and looked great. Despite my general blame-the-graphic-designer philosophy, I had to admit that overall it was a huge improvement.

I went over to mitpress.mit.edu just now (August 10, 1998) to make sure of exactly where in the page Ben had put in the split table. I found that the site has been redesigned a bit since that meeting. Each book page is a navigation header plus one big HTML table so that the user stares at a blank screen while Illustra grinds. Oh well....

Maintaining the Service

What's the bottom line? Site traffic has grown steadily, partly because of new content developed by Terry Ehling, Marney Smyth, and Ken Overton (MIT Press's current Web staff), but also because the catalog information is deeper and more complete. The product catalog turned out to be a failure as a community Web site; only a handful of readers commented on books. There is no obvious technical reason for this failure since the comment facility is similar to that offered by amazon.com, where readers engage in lively debate (check the comments for *Database Backed Web Sites*, at http://www.amazon.com/exec/obidos/ISBN=1562765302). It can't be explained by a total lack of reader interest since nearly 1000 readers per month are signing up to be notified of new books in various categories.

The invisible shopping cart system and sending orders to the legacy system turned out to work just fine. The site has processed many thousands of orders without imposing an undue burden on folks at MIT Press.

The AOLserver/Illustra combination turned out to require minimal maintenance. MIT Press did not have to hire full-time computer science geeks, database administrators, or Unix system administrators. They've been able to concentrate almost all of their efforts on developing new Web content.

Should anyone be impressed by these results?

Yes! We put about four person-weeks of programming into the site and it has the same functional capabilities as amazon.com circa 1997 (amazon had more than 10 full-time programmers). We did not waste server resources with CGI scripts that reopened the database and hence MIT Press did not have to invest $150,000 in a monster Unix box. We did not use an application server or otherwise invest programmer resources in layers of abstraction to insulate the site from Illustra-specific SQL. Consequently, a variety of people without formal computer science backgrounds have been able to edit and adapt the AOLserver Tcl scripts to their needs, extend the administration section, or get me to swoop in and make the occasional surgical change.

It wouldn't be so impressive if there weren't so many folks out there who spend $1 million on infrastructure, packaged software, and software development before they can take their first order.

Could We Do it in 1998 with Packaged Junkware?

There are many more ecommerce and catalog server packages available in 1998 than there were in 1996. Could we build the site now with packaged software? Perhaps, but not very elegantly. Though the MIT Press product line is very easy to characterize formally (books have unique ISBNs and journals have unique ISSNs), there are some screwy items. The user interface would become horribly clunky if there weren't a clean way to present multi-volume sets and relate paper and hardcover editions of the same title. There are also two completely separate categorization systems for books, one for ordinary consumers and one for professors seeking books to use as textbooks.

Each journal needs to have institutional, individual, and student subscription rates. Back issue prices need to be similarly trifold. Finally, there are separate prices for back issues that are "double issues". In a turnkey catalog system, you'd probably have to represent one journal as nine separate products in order to offer these nine separate prices. That would mean MIT Press personnel would have to change journal titles or editorial board info in nine separate database rows.

Because we did not use packaged software, we were able to program in lots of shortcuts on the admin side. The administration home page for a book has the stuff you'd expect from a commercial software package, e.g., links to all the orders for the book and the readers who've expressed interest in the book. But it also has one-click catalog maintenance shortcuts, e.g., "Record this title as being out of print" or "Change the price by editing the old price here and hitting carriage return."

For the Hard-core Nerd

If you're a hard-core nerd, you'll probably be interested to read my annotated data model file: http://www.photo.net/wtr/thebook/mitpress-data-model.txt.

MIT Press Summary

With good people, good management, reasonable goals, and reasonable judgement, we managed to develop, operate, and maintain a moderately high volume and full-featured ecommerce system at minimal cost.

Mundane Details: Running Credit Cards

If you knew nothing about banking, you'd probably think that every bank in the world had a transaction server on a network. If a merchant wanted to figure out whether a credit card was valid or whether a cardholder's billing address was as stated, the merchant would send a packet of information to its favorite bank's server and that bank would come up with the answer, possibly doing queries to other bank's servers.

It turns out that banks never wanted to do anything like this. They left credit card processing to a handful of third-party companies such as First Data (http://www.firstdata.com), NOVA (http://www.novainfo.com), and Paymentech (http://www.paymentech.com). When a consumer's card gets swiped at the supermarket, the little terminal talks to NOVA, for example, and NOVA talks to the consumer's bank and the supermarket's bank.

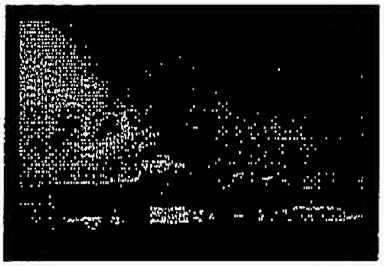
Okay, so now the obvious thing would be for each of these third-party companies to park a transaction server on the Internet. Then a merchant can choose to contact NOVA or Paymentech by sending encrypted packets rather than dialing up. But it apparently turned out that even NOVA and Paymentech didn't want to wade into the public Internet.

Drop a Dime

The most direct and obvious way to bill credit cards is to stick a modem on the back of your Unix machine and have it dial up a credit card processor such as First Data. Your Web site looks to the world's banking system just like the pizza shop around the corner. If you want fast connections, you can get a frame relay or ISDN line to the processor. If you don't want to figure out the protocol that is being used by the little terminal in the pizza shop, you can pick up some software such as HKS's CCVS (see http://www.hks.net) for about $1000.

One downside of this approach is that you'll be forced to maintain a highly secure Web server/database containing customer credit card numbers.

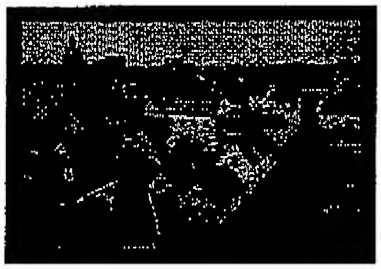
Two Internet Architectures



For people who don't want to stick modems on the backs of their computers, there are companies like CyberCash (http://www.cybercash.com), ICOMS (http://www.icoms.com), and Verifone (http://www.verifone.com) that operate transaction servers that then talk to First Data, NOVA, or Paymentech (that then talk to the banks' computer systems).

CyberCash and ICOMS exemplify two very different Web service architectures. In the CyberCash case, the merchant server collects credit card numbers from consumers. The merchant server then talks to CyberCash to find out if the card is valid and bill the card. This lets the merchant offer the customer options like "Use the same credit card as your last order?" If the merchant becomes unhappy with CyberCash, this architecture lets the merchant switch transaction vendors without consumers ever knowing.

What's the downside of the CyberCash architecture? It imposes a programming and security burden on the merchant. For example, if the merchant's programmers aren't competent, customers may get billed for orders that are never entered or shipped. Alternatively, orders may get entered for customers whose credit cards failed authorization. Customer credit card numbers may reside in the merchant database. So if the merchant's database is compromised, not only can the intruder determine who purchased what, but he will get a big table of credit card numbers as well.



The ICOMS architecture assumes that the merchant is incompetent to write programs. A merchant can only embed "digital offers" (little blocks of HTML) in static .html Web documents. As soon as the customer wants to add something to a shopping basket or place an order, the digital offers bounce the customer over to a transaction server run by ICOMS. ICOMS will eventually notify the merchant server that an order was placed. The big advantage of ICOMS is that you don't have to be constantly paranoid about security. Your customers' credit card number might be compromised, but it will be someone else's fault!

A big disadvantage of the ICOMS architecture is that your overall Web service may become unreliable and slow due to factors beyond the merchant's control. For example, under the CyberCash architecture, even if the merchant's server couldn't reach CyberCash, it could be programmed to continue to take orders and then process them in a batch when CyberCash became reachable again. But with ICOMS, the customer can't even type in a credit card numbers or address if ICOMS is down. Furthermore, it is the ICOMS server that offers customers a "use the

same credit card as before" option. So the consumer becomes tied to the ICOMS network and not to the merchant. Finally, you might not be able to achieve your user interface and customer service pages goals with the ICOMS-style architecture.

If you're tempted to use the ICOMS architecture and let another server handle the collection of payment and shipping information, why not go whole-hog and let another server handle the distribution of sales information as well? If your payment pages don't need a custom flow then maybe your whole shop doesn't need a custom flow. You can go over to store.yahoo.com, configure a shop using a Web browser, and pay them $100 a month to present offers and collect orders for you. Yahoo Store offers an interesting comment on the state of ecommerce:

> "Although Yahoo! Store supports real-time credit card authorizations through Cybercash, **we do not recommend it for most merchants.** ... using Cybercash tends to be more trouble than it is worth.
> -- http://store.yahoo.com/vw/cybercash.html (August 10, 1998)

A Gazillion Vendors

If there are only two architectures, how come there are so many listings in http://www.yahoo.com/Business_and_Economy/Companies/Financial_Services/Transaction_Clearing

Some of these listings are deceptive in that they are really front-ends to more basic services. If you don't want to grapple with CyberCash's CGI scripts, there are a few dozen ISPs and Web development companies who will sell you simpler less-general scripts. However, it is hard to tell from reading their Web sites that what they've developed is a 100-line Perl script and not a complete system to compete with CyberCash or Verifone.

An Annoying Future: The SET Protocol

Since 1996, the credit card companies and banks have been working on the Secure Electronic Transaction (SET) protocol. If you're a practical-minded Web merchant, all you really need to know about SET is that it requires your customers to have a Web browser, credit card, special vWallet software, and a certificate issued by their bank. They will only be able to make purchases from their desktop machine where the vWallet software and certificate are installed. Your competitors will be selling to folks who have the Web browser and credit card. Your competitors' customers will be ordering from their desktop machines, friends' computers, network computers in public libraries, etc.

In the long run, the SET protocol could potentially bring the world closer to the "consumers dealing with merchants dealing with banks as peers on the Internet" model that you'd naively assume was the way everything had always worked. However, if you're a Web publisher, "potentially" and "long run" aren't very interesting ideas. Furthermore, all of the payment APIs to which you can write code in 1998 claim to be SET-compliant. So if the great SET day ever arrives, you might not have to change even one of your scripts.

An Extra Layer of Transactions

Generally you can rely on the relational database management system (RDBMS) sitting behind your Web service to handle transactions. You can tell Oracle to "change all of these things and if any of them fail, then undo all the changes and insertions that suceeded." Oracle is extremely reliable at rolling back

transactions but your Oracle system isn't capable of rolling back a change on a non-cooperating foreign system, e.g., CyberCash's server or the First Data computer system into which your Web server dialed.

Suppose that you decide to implement the following architecture:

1. Check data from order form.
2. Make a request to CyberCash (or make a phone call to First Data) to authorize credit card.
3. If CyberCash says "authorized", insert a row into your orders table.

Suppose that someone pulls the power plug on your server right after the request to CyberCash gets sent but before your program has had a chance to receive the result or insert anything into your local database. You would have billed someone's card for $X and yet have no record of the order. The way to get around this is to

1. Check data from order form.
2. Insert a row into your orders table with a status of "confirmed".
3. Make a request to CyberCash to authorize credit card.
4. If CyberCash says "authorized" then update the row in the orders table with a status of "authorized" or, if Cybercash doesn't respond or returns "failed", then update the row with a status of "failed".

Someone can still pull the plug in the middle of these steps, leaving you with a confirmed order in your database and no idea whether or not CyberCash authorized the credit card transaction. So you need a process that runs every hour or so to look for confirmed orders that are older than, say, 30 minutes. The clean-up process will then try to connect to CyberCash and see whether or not the order was authorized. If so, the status is updated to "authorized". If CyberCash has no record of the order, the order can be retried (assuming you elected to save credit card numbers) or pushed into the "failed" state.

Reload 5 Times = 5 Orders?

The obvious way to implement an ecommerce system is the following:

1. Serve user a static order form pointing to an "insert-order" page.
2. When the user hits submit, the insert-order page will run and
   - generate a unique order ID
   - insert a row in the database
   - run the credit card
   - update the row in the database to say "authorization succeeded"
   - print a thank-you page back to the user's browser
   - send email thanking the user for the order

The problem with this approach is that various aspects of Step 2 may be slow, prompting the user to hit Reload. At which point another unique order ID is generated and another row is inserted into the database and the credit card number is authorized again. If the user hits Reload five times, it looks to the merchant's database just the same as five actual orders. The merchant can only hope that the user will notice the duplicate email messages and alert the merchant's customer service

department.

There are potentially many ways of getting around this problem, but I think the simplest approach instead is to

1. Serve user a dynamically generated order form that includes a unique order ID, pointing to an "insert-order" page.
2. When the user hits submit, the insert-order page will run and
   - insert a row into a database table with a primary key constraint on the order id. If the insert fails (Oracle won't allow two rows with the same primary key), catch it and look to see if there is already an order in the database with the same id. If so, serve the user an order status page instead.
   - if the insert succeeded, proceed as above

Eve Andersson, Jin Choi, and I distribute some example software that implements this approach from http://arsdigita.com/free-tools/shoppe.html.

Case Study: ArsDigita, LLC

A Fortune 500 clothing manufacturer came to my friends and me (ArsDigita, LLC) and asked us to build them an ecommerce site. Consumers would type in some personal data, the order would get sent to a factory, a custom-made item would get shipped to the consumer, and the customer's credit card would be billed.

Our response? Too boring. We specialize in community Web sites and grand public services like www.scorecard.org that satisfy our bloated egos when the sites are featured on network TV news.

"We want you to process a few million orders in the next two years. At around $50 each. We have the budget to make this work."

Oh, well, gee, that sounds like an interesting challenge after all...

No modem

We decided early on that we didn't want to talk directly to the card processors via modem. We needed to have a reliable IP connection to talk to Web customers so why not use that to talk to the banking system as well? We'd have to engineer defensively to handle the times when the Internet gateway to the credit card system was unavailable, but we'd also be eliminating two potential points of failure: the modem and the phone line.

Another motivation behind this decision was that we expected to be able to avoid keeping cleartext credit card numbers in our database if we used one of these fancy Internet commerce gateway services.

Deciding between ICOMS-style and CyberCash-style

Given that we weren't going to bring in a phone line, we had to pick between the ICOMs-style architecture and the CyberCash-style system. Our Fortune 500 client was picky about the customer experience and that necessitated the customer talking only to our server, which would in turn talk to the banking network. I.e., we decided on the CyberCash-style architecture.

We looked at the vPOS system from VeriFone and rejected it immediately because it only works with Microsoft Internet Information Server (we don't know anyone who understands NT well enough to run a Web service from Windows NT), Netscape Enterprise Server (we aren't

willing to suffer with Netscape's programming tools, such as LiveWire), and Oracle Web Server (our code library is in AOLserver Tcl).

We looked at http://www.cybercash.com. It offers three options for building "your on-line store":

1. Hire an officially sanctioned programming team to build you a site.
2. Buy some packaged on-line store junkware from a collection of officially sanctioned vendors.
3. Download their core C and Perl software (the CyberCash API), plus some unsupported examples that use the core API calls, then start integrating them into "your existing on-line store".

We don't even like to look at our own source code sometimes, much less anyone else's. So that ruled out Option 1. We thought it would take longer to read the marketing literature for all the packaged junkware than it would to just write everything from scratch, so that ruled out Option 2. We chose Option 3 and registered to download the Merchant Connection Kit 3.2.

Due to a combination of bad programming and user interface design on the CyberCash site plus sluggish customer service personnel, we weren't able to even download the software for 48 hours. The downloaded kit was a gzipped Unix tar file, but it bore no extension so it took a bit of Unix expertise to (1) figure out what kind of file it was, and (2) rename it to foobar.tar.gz and gunzip it.

At this point, we had to decide whether to use the Perl interface or the C library. In general, we prefer to use interpreted languages for Web software development. However, since we were building the entire site in AOLserver Tcl, it seemed tasteless to maintain some of the pages in Perl CGI. Jin spent a night writing a 200-line C program (see http://www.photo.net/wtr/thebook/cybercash.c).

The CyberCash API consists of exchanging key-value pairs. There is really only one basic C procedure that one needs to call and Jin's code is a bridge between the CyberCash key-value data structure and AOLserver's `ns_set` data structure. All the interesting information about the API is in Appendix B of http://www.cybercash.com/cybercash/merchants/docs/dev.pdf.

To the extent that CyberCash is tricky, it is mostly because the credit card world is archaic and tricky. There are two fundamental steps in processing a credit card order. The first step is authorization. The merchant gives the processor a card number, expiration date, name on card, street address, and zip code. The processor is fundamentally checking that the card number is valid and that the cardholder has sufficient credit to handle the amount authorized. Given the religious zeal with which merchants collect your card expiration dates, you'd think that processors did something clever with this. In fact all they do is make sure that it is a date in the future. So if a consumer's card expires in 06/00 and he gives you 01/00, First Data will not reject the transaction. The address verification service (see below) is purely to help merchants assess the risk of sending out a product and then later finding out that the credit card number was purloined. The processor says "Zip code matched but address did not" or "Nothing matched". The merchant has to decide whether or not to take the risk of shipping the good.

The second step in processing a credit card order is settlement or "capture." With a hard good, the merchant would typically authorize when it gets an order and capture when it ships the product. With a soft good, e.g., the sale of access to a Web service or document, the merchant would typically capture immediately after authorization.

Complicating matters somewhat is the fact that processors try to conserve their network and server resources by coercing merchants into performing settlements in batches of five or more. A merchant can kick CyberCash into autosettlement mode whereby Cybercash periodically sweeps its database to find transactions that are authorized *and* marked for settlement. So the hard goods merchant's two basic operations become "mauthonly" (to authorize; done when order is received)

and "postauth" (to mark for settlement; done when good is shipped). A soft goods merchant can put Cybercash into auto-mark, auto-settle mode whereby authorized transactions are immediately marked for settlement.

Our honeymoon with CyberCash

What's great about CyberCash is that it is so centered on the merchant's order ID. We give CyberCash a card number and a unique order ID and from then on we can do all of our transactions merely by referencing this order ID. These transactions include "void" (halt the settlement process), "return" (refund a consumer's money for a settled transaction), and "card-query" (get the full cleartext card number back for a particular order).

Another nice thing about Cybercash is that they have a full Web site where a merchant can go to review all the orders, do refunds, run cards one at a time, etc.

The end of the honeymoon?

Our brilliant scheme would founder if Cybercash did not keep order data around for a long time. It might take a customer 10 days to receive a good and 45 days to decide that he didn't like it and send it back. We don't want to have to ask him for his card number again. If it took us more than seven days to mark an authorized transaction for settlement then the cardholder's bank might reject the settlement and we'd have to reauthorize (using the card number). So we absolutely depend on CyberCash to keep the card numbers around for a fairly long time in *their* database. Currently they state privately that they will keep these data for 90 days, which seems long enough for most purposes.

That leaves one big remaining hole: If CyberCash were unreachable or could not reach the card processor or the card processor were down, we'd have to either bounce the customer or keep his card number and retry later.

What would plug this last hole is for CyberCash to give us a public key. We'd encrypt the consumer's card number immediately upon receipt and stuff it into our Oracle database. Then if we needed to retry an authorization, we'd simply send CyberCash a message with the encrypted card number. They would decrypt the card number with their private key and process the transaction normally. If a cracker broke into our server, the handful of credit card numbers in our database would be unreadable without Cybercash's private key. The same sort of architecture would let us do reorders or returns six months after an order.

CyberCash has "no plans" to do anything like this.

Disputed Charges

Disputed charges or chargebacks are a much greater risk in the mail-order telephone-order (MOTO) world than in the pizza shop face-to-face world. Many banks don't even like to give merchant accounts to companies who say they want to take telephone or Internet orders. What happens is that 45 days after the

transaction is authorized, the cardholder writes to his bank and says "I didn't receive my goods" or "I don't know what this charge is for." The merchant finds out about this in a report from his bank; there is no indication of a charge dispute in the CyberCash database.

What particularly scares a bank about this is that they've already given the merchant the money. Consider the case of a Times Square camera shop. They take $1 million in credit card orders for cameras that they promise to ship in four weeks. The money gets piped into their bank account immediately. They take the money to Brazil. Two months later, their merchant bank is forced to refund $1 million to various cardholders' banks, with no recourse to the merchant.

Before inflicting our software on the Fortune 500 company, we decided to open a test merchant account through which we'd run charitable contributions to a couple of animal shelters (see http://www.photo.net/samantha/gift-shop.html). We thought it would be easy to get an account from BankBoston. They'd recently discovered that Ricardo S. Carrasco, one of their senior lending officers, had written $73 million in fraudulent loans in Latin America and then disappeared. We therefore figured that ArsDigita would look great by comparison. We'd been customers since 1979, had $175,000 on deposit, and no connections to Argentine businessmen with "histories of criminal and financial problems."

It turned out that we weren't quite BankBoston's dream customer as we'd envisioned but five weeks later, we had our merchant account. If you're setting up a site, it is probably unwise to rely on service faster than that from a typical bank.

Fees

Getting rich on the Internet isn't cheap. Here's what it cost us to get set up with BankBoston plus Cybercash:

- $375 to set up a merchant account with BankBoston (plus it will cost us another $100 to terminate our account)
- $40 a month to BankBoston or 4.8 percent of the total transactions, whichever is larger
- $595 setup fee to Cybercash
- $40 a month to Cybercash *plus* 20 cents per authorization

Compounding the horror of these payments is the fact that we have three separate companies with which to deal: BankBoston (our bank), First Data (their card processor), CyberCash (our gateway to First Data). If there is a problem with our merchant account, we have to first figure out whose fault it might be and who might have the expertise to fix the problem.

Sales Tax

Our Fortune 500 client has "nexus" in all 50 states, meaning that in each state they are considered to be "engaged in business" and therefore required by law to collect sales tax. Note that they weren't actually selling goods direct-to-consumer nationwide, but had warehouses and salespeople in every state.

This doesn't sound so bad, does it? We just need an Oracle table with 50 rows, each one containing the tax rate for a particular state. Actually it turns out that there are about 7,000 taxing jurisdictions in the United States and 17,000 different tax rates. Okay, that's no problem either. Oracle can handle a 17,000-row table quite easily. We can buy the data for $3,000 a year from http://www.salestax.com. A few straightforward Tcl scripts and

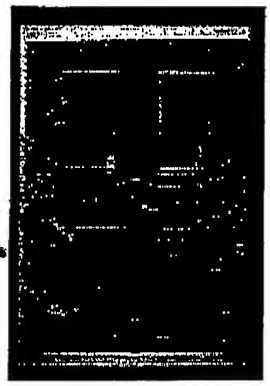
SQL queries and we can compute the amount of tax to collect based on the zip code of the shipping address.

So what's the problem?

We've *collected* the tax. We haven't *paid* the tax. We now need to write checks and fill out forms for those 7,000 taxing jurisdictions nationwide. Most states let you remit your local jurisdiction taxes to the state along with a schedule and then the state revenue folks distribute the appropriate amounts periodically to local governments. Still, assuming that our sales are initially light, we could be writing one or two checks and filling out one or two forms for every product that we sell. We will be hiring a big staff of people to sit in front of a forms CD-ROM from http://www.salestax.com or a more automated software package from http://www.corptax.com, http://www.taxware.com, or http://www.vertexinc.com.

The interesting thing to note here is that big companies that have always sold wholesale may initially see the Internet as a great opportunity to cut through layers and get directly to the consumer. Yet the cost of building a sales tax compliance department may wipe out many years of profits from Internet sales.

Accounting

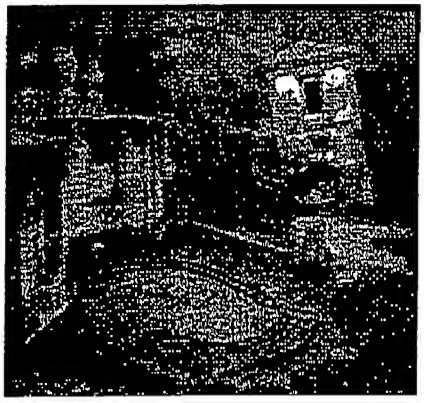


Periodically, we have to tell the accounting folks what happened on the Web site so that they can finish the company's financial statements and taxes. You'd think that an easy way to do this would be just to hand over the merchant bank's monthly statements showing the flux of money going to and from consumers' credit cards. That's sort of like the cash-basis accounting that individuals and very small companies do. Large companies, though, are required to use *accrual basis*. They are supposed to recognize revenue when they know that they're going to get it, rather than when they actually get it.

Should we thus build accounting reports that show orders when they were confirmed by a consumer? No. It turns out that this makes accounting way too hard because assets can be double-counted. For example, the consumer on December 30, 1998 says he wants to buy a widget for $100. You ship the widget on January 3, 1999. At the end of 1998, your books will show that you have a widget in inventory, value $90, *plus* an order for the same widget, value $100.

Companies get around this problem by recognizing revenue when the product ships. The widget gets removed from inventory and the order gets added to revenue. There is never a risk of double counting.

ArsDigita Shoppe



We decided that we'd have to go live with a full-scale charity shop at least one month before going live with the "big custom clothing shop". Our first product was a $20 donation to Angell Memorial Animal Hospital, a local veterinary research center. People who contributed $20 would receive a pair of prints from my Italian collection (http://www.photo.net/italy/). Ever since the inception of my site, I'd been giving away prints to people who donated money but the prices were higher ($100 to $250) and donors had to put a check in the mail. However, for the first time we would have a link right at the top of photo.net, a price suited to impulse purchasing, and on-line ordering.

For the charity test, we used our "ArsDigita Shoppe" product, a comprehensibly small set of AOLserver Tcl scripts that merchants are expected to customize. There are only a handful of database tables in the system. Note that they are all prefixed with "sh_" so that they can be installed in an existing Oracle database without much chance of name conflicts with previously defined tables. Here are the basics of the data model:

- `sh_products`, containing product descriptions and prices, keyed by `product_id`
- `sh_orders`, containing customer information such as shipping address, whether or not the order was shipped, and records of conversations with CyberCash
- `sh_problems_log`, to record transactions where CyberCash failed or did not return
- `sh_country_codes`; used to enforce an integrity constraint on the country field of `sh_orders`

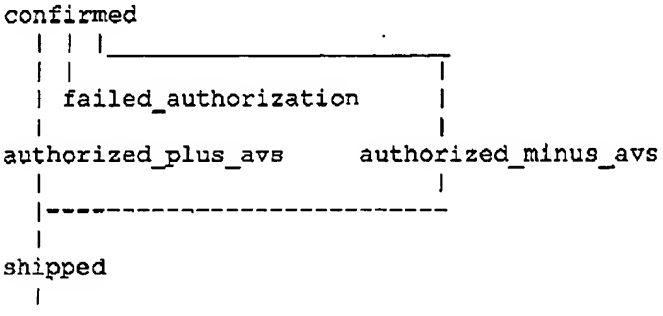
ArsDigita Shoppe is designed for maximum simplicity and security. It does not ever store credit card numbers and must therefore reject any order that cannot be immediately authorized through CyberCash.

We decided that we'd build the system assuming the merchant had put CyberCash into "auto-settle, manual-mark" mode. For products flagged as soft goods in the `sh_products` table, we'd do an "mauthonly" and then an immediate "postauth". For regular products, we'd have a fulfillment section among the admin pages where whoever was doing shipping could mark orders as shipped when they went out the door. The order status would be updated to shipped and CyberCash would be sent a "postauth".

The finite-state machine for orders

For both customer service and accounting, a merchant needs to know whether an order has been paid for, shipped, returned, etc. As far as I can tell, the world of business software operates either via recording lines in tabular ledgers or via messages sent from one "system" to another. So you can tell whether or not an order had been shipped depending on whether or not there are rows with that order_id in particular ledgers or in particular systems.

Though I'm sure the ledger and message-based approaches feel natural to people in the business data processing world, they feel awkward to me. I visualize an order as being in one of a finite number of states. For example, `confirmed` means the customer has pressed the submit button and indicated a desire to place the order. From `confirmed`, it is possible to get to one of three states: `failed_authorization`, `authorized_plus_avs`, and `authorized_minus_avs`. Whether a state transition occurs and to which new state depends on what our software gets back from CyberCash. Here's a graphical representation of the finite-state machine (FSM):



```
   |
returned
   |
   |
refunded

Note:  the order can also be  put into the state "void" by the shoppe
administrator from any state prior to "shipped"
```

An authorized order is put into the `shipped` state when the merchant clicks on the admin fulfillment form's "confirm that this has been shipped" box. As these transitions occur, we are simply updating the `order_state` column in the `sh_orders` table. That doesn't mean we're losing information versus the ledger approach. There are extra columns in the `sh_orders` table to hold the data that accumulate upon state transitions.

For example, after the Tcl order handling procedure talks to CyberCash to do the initial authorization, it records the CyberCash transaction ID, authorization code from the credit card processor and Address Verification Service code returned by CyberCash.

Address Verification Service

The credit card processing system was designed to quickly figure out "Is this a currently valid credit card and does it have enough available credit to handle this purchase?" Through CyberCash and First Data, we've had no trouble getting authorizations for orders when we supplied the wrong name on the card or the wrong expiration date. This presumably isn't too bad of a problem in the face-to-face world. The cardholder has physical possession of the card. If he loses it, he will probably call the bank and have it canceled. So if a merchant sees a customer waving a card, it is almost certain to be the cardholder. However, in the mail-order telephone-order world, the merchant never has the opportunity to verify that the customer has physical possession of the card, only that the customer knows the number of a valid credit card. The number could have been obtained in many ways without the cardholder ever knowing.

If a merchant ships a $1000 widget to a crook and the real cardholder complains, the merchant has to give $1000 back to the bank. In order to help mail-order telephone-order merchants manage this risk, the card processors run what seems to be an essentially separate system called Address Verification Service. They take the first 20 characters of the street address and the billing zip code and return a one-character code:

- Y means "five-digit zip code and address both match"
- A means "address matches, zip code does not"
- Z means "zip code matches, address does not"
- N means "neither address nor zip code matches"
- R means "system unavailable or timed out"
- S means "card type not supported"
- ...

As a merchant, you have to decide what to do in these various cases. ArsDigita Shoppe encapsulates this business decision in a single procedure containing a list of codes that are deemed sufficient to put an order into `authorized_plus_avs`. The merchant also needs to make a business decision for every order that lands into `authorized_minus_avs`. To ship or not to ship? That's the question. ArsDigita Shoppe by default will solicit a customer's telephone number and

presumably the merchant will be advised to email or telephone the customer for verification.

Note that in our experience AVS is not very reliable. For our first few successful orders, all of which were from legit folks, here are our stats:

AVS Code	Count
N	2
R	1
S	1
Y	11
Z	28

The 11 cases in which both address and five-digit zip codes match looks pretty good until you look at our source code and note that we neither solicit the card billing street address from the customer nor do we ever provide any street address to CyberCash. We only solicit card billing zip from the customer and send that through to CyberCash. Consequently, the 28 cases of "zip code matches" are to be expected. In two cases, the AVS code raises an unreasonable suspicion ("no match"), and in two more cases we couldn't get an AVS code at all. The one "card type not supported" (S) was a friend's Mastercard. Oddly enough, it was issued by BankBoston, our merchant bank.

Integrity

We assume that the following components could fail at any instant:

- our server hardware or software
- our communication with CyberCash
- CyberCash's servers
- CyberCash's communication with the card processor (First Data)
- the card processor's network or systems

Later in this section, I will discuss the software that we built to recover from failures on the above list. Meanwhile, we assume that the following components are 100 percent reliable:

- transactions committed to our Oracle database
- orders recorded by CyberCash

The consequences of the merchant's Oracle database losing a transaction might include the merchant billing a customer and never knowing that product needed to be shipped or the merchant shipping product to the same customer twice. Recall that ArsDigita Shoppe does not store credit card numbers. Therefore, if CyberCash lost an already-shipped order, the merchant would be unable to issue a refund if the customer returned a product. If CyberCash lost a to-be-shipped order, the merchant would be unable to mark the order for settlement and therefore would not be able to fulfill the order. If CyberCash lost a shipped and marked for settlement but still -to-be-

settled order, the merchant would have already shipped the product and would not get any money for it.

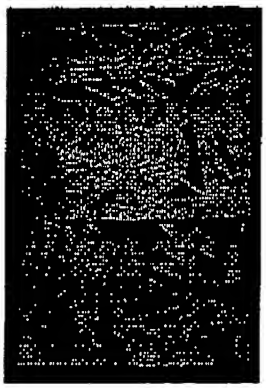
I've written a little bit elsewhere in this book about how to make an Oracle installation that never loses committed transactions, even if a disk drive fails. Suffice it to say that you probably need at least eight physical disk drives plus a full-time staff person who has read every book on my Oracle bookshelf (http://www.photo.net/wtr/bookshelf.html).

How to make sure that CyberCash never loses committed orders? This is something you'd hope you won't ever have to do. CyberCash runs a database server and therefore is subject to the same db/sysadmin challenges as everyone else. I haven't talked to enough of CyberCash's internal people to say whether I believe that they'll never screw up their database administration.

ArsDigita Shoppe defends against failures to communicate with CyberCash by periodically running clean-up processes looking for orders in intermediate states. Scheduled by AOLserver's `ns_schedule_proc` (like the Unix cron facility, but schedules a procedure to run inside AOLserver where it has access to all the AOLserver API calls), we have the following procedures:

- `sh_sweep_for_cybercash_zombies`. Runs every hour. Looks for orders in the `confirmed` state that are older than 30 minutes. Queries CyberCash to find out whether or not an authorization request was received and, if so, whether it succeeded. If CyberCash is reachable, the procedure will always push the order either into `failed_authorization` or one of the authorized states.
- `sh_sweep_for_unmarked_shipped_orders`. Runs every hour. Looks for orders in the `shipped` state that are older than 30 minutes but for which we have not recorded a "postauth status" code back from CyberCash (i.e., those orders that should be marked for settlement but might not be). Queries CyberCash to find out whether or not order is marked. If not, the procedure will try again to mark it.
- `sh_query_for_settled_and_setlret`. Runs at 3:14 am every day (if the time seems strange, see Eve Andersson's site at http://eveander.com). By 3:14 eastern time, CyberCash theoretically should have settled all the previous day's marked orders. This procedure grabs those settlement codes and updates the `sh_orders` table.

Going Live



We launched our charity shop at around 2:00 am on Thursday, June 18, 1998. Here is our experience to date:

- 3:45 am, June 18: Matthew Haughey of UCLA is our first customer, donating $20 to Angell Memorial; we have a total of 24 successfully authorized orders by the end of the day.
- 2:37 am, June 19: CyberCash begins rejecting authorization for legitimate cards, reporting "invalid card number" to the customer. We send email to merchant-support@cybercash.com; it has yet to be answered.
- mid-day, June 19: CyberCash is unreachable. Their API returns "Could not open socket to connect to Merchant Payment Server." It is impossible to ping or traceroute to cr.cybercash.com (the host that processes payment requests and also lets a merchant log in). Nearly 20 orders are rejected due to inability to communicate with CyberCash. We send email to merchant-support@cybercash.com; it has yet to be answered.
- 8:30 pm, June 20: We are trying to fulfill the 33 successful orders in our database.

CyberCash fails our requests to mark the orders for settlement saying either "PostAuth: There is no previous authorization for this order" or "DataBase error #-901: System error, RDM runtime memory may be corrupted." We got to https://cr.cybercash.com and log into the Merchant maintenance pages. All of the orders we'd received on June 18th and 19th have disappeared from CyberCash's records.

- 10 pm, June 20: CyberCash fails to process two orders, returning "DataBase error #-901: System error, RDM runtime memory may be corrupted." *(I later learned that RDM stands for "Raima Database Manager", a system that is cheaper than Oracle but apparently not bulletproof. CyberCash says that they are moving to Oracle.)*
- 11:48 pm, Sunday, June 21: https://cr.cybercash.com is reachable again but all of our orders from June 18th through June 20th are missing. We telephone CyberCash merchant support at (703) 295-0888. They're closed but we push 0 to wake up an on-call support guy. We supply him with the transaction IDs (given to us by CyberCash) of a few of the authorized-by-them-and-now-lost orders. He says that it looks like we'll need "technical support" and they're not available until Monday morning.
- 2:30 pm, June 22: Gale from CyberCash calls to say that she expects the missing data to be restored by the morning of the 23rd.
- 10:23 am, June 23: A couple of orders fail from CyberCash, with "failure-q-or-cancel; Premature EOF read from socket. HTTP body is missing. A likely reason: server did not respond or dropped a connection". Our software is programmed to retry these once but apparently that was not enough.
- 1:30 pm, June 23: Our orders are still missing. No word from CyberCash.
- 4:40 pm, June 23: Eve telephones Greg at CyberCash to find out why our Fortune 500 customer's tests were failing earlier:

  E: Why did this valid credit card fail with error message "Invalid Credit Card Number" today between 10:30 and 10:49 am, even though it has since been tested and found to be valid?
  G: It could be because of Internet traffic.

  E: Why would that cause the message "Invalid Credit Card Number"?
  G: Maybe it got garbled in transmission.

  E: How could it get garbled 12 times?
  G: Maybe the system was down. Let me check... Yes, it was down shortly before 11 this morning.

- 1:30 pm, June 24: Our orders from June 18th through June 20th have reappeared. The card types are missing from standard reports and the AVS codes have been garbled (they no longer fit into our `char(3)` Oracle column), but we're able to settle transactions.
- 6 pm, June 29: Our first customer of the day (sales at our charity shop have slowed!) tries to place an order but our machine can't reach CyberCash so the customer gets hit with a "cannot connect to payment server" message. We start testing https://cr.cybercash.com and it responds, albeit very slowly. The customer retries the order five minutes later and succeeds. I call customer support and, after keeping me on hold for 10 minutes, they verify that their payment server was down, but can't say why. They very helpfully give me a service status page: http://merchant.cybercash.com/cgi-bin/status/cyber/status.cgi. This page doesn't give a cumulative history, however, for CyberCash or any of its card processors.
- 9 pm, June 30: We take a closer look at the CyberCash status page. The data on the page appear to be incomplete. For example, the payment server's outage on June 29th is not shown. The last outage is instead listed as June 26th. Their last problem with Internet

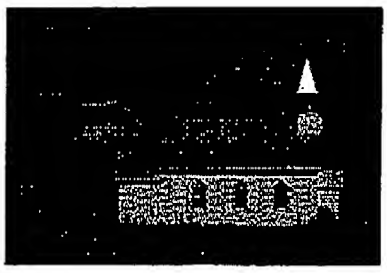
connectivity is shown as having been on February 19th. Yet we were unable to reach CyberCash on May 18th and narrowed the problem down to a routing loop (either their fault or their ISPs):

```
www: ~% traceroute cr.cybercash.com
traceroute to hcr2.cybercash.com (204.178.186.35), 30 hops max, 40 byte
 1  main-98.sjc.above.net (207.126.98.4)  6.86 ms  14.535 ms  20.903 ms
 2  main-98.sjc.above.net (207.126.98.4)  1.336 ms  1.209 ms  1.243 ms
 3  paix-main-oc3.above.net (207.126.96.122)  5.883 ms mae-east-paix.abc
 4  Hssi0-1-0.GW1.TCO1.ALTER.NET (157.130.33.113)  76.284 ms  70.12 ms
 5  421.atm10-0-0.cr2.tco1.alter.net (137.39.13.6)  71.226 ms  70.655 ms
 6  Fddi0-0.Vienna3.VA.Alter.Net (137.39.11.4)  78.618 ms  164.081 ms  8
 7  CyberCash-gw.customer.ALTER.NET (137.39.154.226)  98.223 ms  74.49 r
 8  204.149.69.253 (204.149.69.253)  79.065 ms  80.27 ms  75.728 ms
 9  cust-gw.cybercash.com (204.178.187.250)  80.119 ms  81.686 ms  79.3
10  ans-gw.cybercash.com (204.178.187.253)  84.779 ms  85.597 ms  83.99
11  cust-gw.cybercash.com (204.178.187.250)  83.425 ms  81.273 ms  88.2
12  ans-gw.cybercash.com (204.178.187.253)  87.992 ms  87.012 ms  89.31
13  cust-gw.cybercash.com (204.178.187.250)  84.027 ms  84.861 ms  85.0
14  ans-gw.cybercash.com (204.178.187.253)  90.975 ms  90.937 ms  87.99
15  cust-gw.cybercash.com (204.178.187.250)  85.138 ms  90.371 ms  84.6
16  ans-gw.cybercash.com (204.178.187.253)  93.612 ms  93.498 ms  93.07
```

- 3 pm, July 8: Fulfilling a few orders, we get hit with "Could not connect to Merchant Payment Server" on one but the others succeed.
- August 1998: CyberCash seems to have settled down and is working fairly reliably for both ArsDigita Shoppe and our Fortune 500 customer. Our biggest problem is the confusion generated by the reports that we get from BankBoston. Money seems to get dumped into our bank account and then, about half the time, taken back the next day. It isn't clear whether this is a First Data/BankBoston problem or a CyberCash/First Data problem. In any case, reconciling these reports with our order database would require a team of five Talmud scholars.

Some of the publishers who use the ecommerce module for the ArsDigita Community System (see http://www.photo.net/doc/ecommerce.html) use CyberSource rather than CyberCash. CyberSource has some nice extras such as calculation of sales tax. What about reliability? The December 22, 1999 *PC Week* cover story is "CyberSource server outage strains e-tailers". The best part of the article, chronicling a 7-hour outage, is the report that William Donahoo, VP Marketing at CyberSource, "promised such an outage would never happen again."

ArsDigita Summary



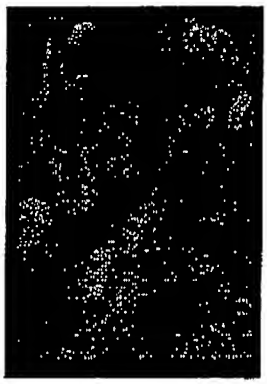
So far, we've learned that, if you aren't willing to keep credit card numbers temporarily, you lose orders while your means of billing credit cards is unavailable. We've also seen that, if you aren't willing to keep credit cards permanently, you had better pray that CyberCash keeps its Oracle database together.

For our Fortune 500 client, we built a fancy system that is robust to CyberCash failures. If CyberCash is unreachable or returns an inconclusive answer, we assume that the credit card number is valid and have clean-up cron jobs that keep trying to authorize the card in hopes that CyberCash has come back to life. This code turns out to be pretty complicated, especially since it has to deal with the case of "that guy who placed an order 18

hours ago is over his credit limit but we didn't find out until just now because CyberCash was down."

An alternative architecture that we haven't explored would rely on at least two IP-based services like CyberCash. When a new order came in, we'd try to authorize via Credit Card Gateway A. If it was unreachable or inconclusive, we'd try to authorize via Credit Card Gateway B.

Something Interesting



Now that I've bored you to death with all of that ecommerce stuff, is it possible that there is anything interesting here? I think so. Karl Marx may not have been an accurate prophet but he remains the world's greatest analyst of the industrial revolution. Why was the factory worker's life so miserable? One reason was that the worker never got to talk to a customer, never got to see how his work was used.

Can we fix all of the ills of the modern age with ecommerce? Probably not. In fact, it seems likely that ecommerce will add to the discontents of civilization ("we already killed Downtown; now let's go after the Mall"). But I think it is possible to use technology to ameliorate the problem identified by Marx. A company can use its Web site to let consumers talk to the people who make the products and vice versa.

Instead of working an eight-hour factory floor shift, why can't a worker spend six hours a day building products, one hour a day answering customer tech support email, and one hour a day answering sales inquiries? For semi-custom goods, why can't the people building the product answer the "When is it going to be done" questions? Or directly ask the "Are you sure you want it with two screws on top because it would look much nicer fastened from the edges" questions?

Note that this kind of direct customer-worker contact could become a competitive advantage for companies manufacturing domestically. Could I communicate effectively with the 14-year-old Indonesian girl who made my Nike sneakers? Perhaps not. But my Toyota Sienna minivan was built in Kentucky. I'd love to e-mail the folks who built it a picture of Alex in the driver's seat.

Summary

Ecommerce is at once mind-numbingly boring and terrifyingly difficult. It turns out to be a miracle that most businesses operate successfully at all. With ecommerce, people start demanding that all the business systems that have been up 8 hours a day, 5 days a week with human intervention now be available 24x7 and be fully automated. The worst thing about building ecommerce systems is that nobody will notice if you do the job right.

More

- The source code for ArsDigita Shoppe may be obtained from http://arsdigita.com/free-tools/shoppe.html.
- If you want to run an integrated online community and ecommerce system, the ArsDigita Community System contains an ecommerce module, described in http://www.photo.net/doc/ecommerce.html.
- For a look at how Microsoft has tried to solve some of the same problems, visit http://www.microsoft.com/siteserver/ or get *Microsoft Site Server 3.0 Bible* (Harris et al 1998; IDG).
- If you're working in this area, it is useful to understand business data processing software,

the most successful example of which is SAP (www.sap.com). Popular books on SAP include *Sap R/3 Business Blueprint : Understanding the Business Process Reference Model* (Curren et al 1997; Prentice Hall) and *The Sap R/3 Handbook* (Jose Antonio Hernandez 1997; McGraw-Hill).

Get Hardcopy or move on to Chapter 15: Case Studies

---

philg@mit.edu

Related Links

- All in one mall- shopping electronics, cars, clothing, apparel, collectibles, sporting goods, digital cameras,the world's online All in one Mall, book,games,car, apparel,collectibles, sporting goods & more at low prices, Buy your own e-store,All in one on axlstore.com. (contributed by oliver fridman)

- ImGlobal - Merchant Account / Online Credit Card Processing- ImGlobal is a Merchant Services Provider for businesses that require credit & debit card processing services to accept payments for Internet, Mail-Order Telephone-Order (MOTO), & POS/Retail transactions. ImGlobal Offers Payment Solutions that Lead Your Business to The Web, Safely. (contributed by Ben Felczer)

Add a comment | Add a link